# Introduction to NetLogo

Advanced Methods of Modelling and Simulation 2013/2014

# Outline

- Introduction to NetLogo
  - Turtles, Patches, and others
  - Interface
  - Programming Concepts
  - Skeleton set-up
  - Some about plotting in NetLogo
  - Extensions & Tools
- Some examples

2

# Introduction to NetLogo

## What is NetLogo

- A programmable modelling environment for simulating natural and social phenomena (Uri Winlensky 1999)
- Agent-based M&S tool
- Well suited for modelling complex systems
- Hundreds or thousands of independent agents operating concurrently
- Exploring the connection between the micro-level behaviour of individuals and the macro-level patterns that emerge from the interaction of many individuals

3

# Introduction to NetLogo

## What is NetLogo

- Easy-to-use application development environment
- Quickly testing hypotheses about self-organized systems
  - Open simulations and play with them
- Large collection of pre-written simulations in natural and social sciences that can be used and modified
- Simple scripting language
- User-friendly graphical interface

4

# Introduction to NetLogo

## The World of NetLogo

- NetLogo consists of agents living in a 2-D world divided into a grid of patches
- Three different type of agents plus one more
  - **Turtles**, are the agents that move around the world
  - **Patches**, are the pieces of “ground” on which turtles can move
  - **Links**, are agents that connect two turtles
  - **Observer**, is an agent without location that observes everything going on in the world.

5

# Introduction to NetLogo

## NetLogo Components

- Important concepts:
  - All the components can be seen as agents.
  - They can have their own properties, can be given commands, can detect and interact with other agents in their environment.
  - The simulation model is controlled by the observer.
  - The definition of agent behaviours is defined by the observer by using the ask command.

6

# Introduction to NetLogo

## Patches, Turtles, System

- Patches: Elements of space
  - Change
  - Do not move
- Turtles: “Social” actors
  - Change
  - Mobile
- All turtles and patches put together
  - Typically, we wish to observe the system
  - How many turtles are sick? Alive?

7

# Introduction to NetLogo

## “Rules”

- Turtles and patches have rules that can
  - Change themselves (reflexive)
  - Change other turtles
  - Change other patches

8

# Introduction to NetLogo

## Rules for Turtles

- Reflexive behaviour
  - ask turtles [ forward 1 ]
- Reflexive state
  - ask turtles
    [ if (sick?) [ set color blue ] ]
- Change other turtles
  - If (sick?) [ ask turtles here [ set sick? true
    set color blue] ]
- Change patches
  - ask turtles if (sick?)
    [ ask patch-here [ set grass grass – 5 ]]

9

# Introduction to NetLogo

## Rules for Patches

- Reflexive state: patches change themselves
  - ask patches [set grass grass + 1 ]
- Change other patches
  - ask patches in-radius 1 [ set grass 0.1 * my-grass ]
- Change turtles
  - ask turtles-here [ set sick? true
    set color blue ]

10

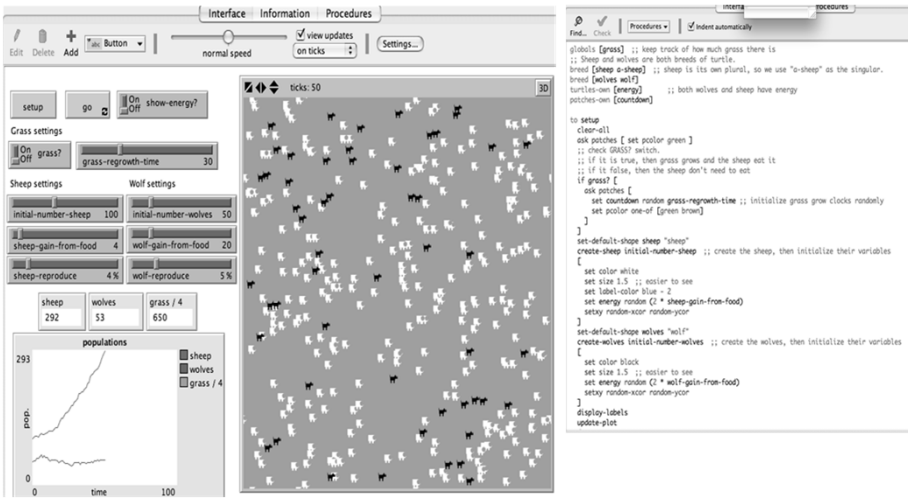
# in Summary

- Tself
- Pself
- T-to-T
- P-to-P
- T-to-P
- P-to-T

11

# Introduction to NetLogo

## GUI - Controls, Settings, Views




12

# Introduction to NetLogo

## GUI - Controls, Settings, Views

- **controls** (BLUE) - allow to run and control the flow of execution
  - buttons
  - command centre
- **settings** (GREEN) - allow to modify parameters
  - sliders
  - switches
  - choosers
- **views** (BEIGE) - allow to display information
  - monitors
  - plots
  - output text areas
  - graphics window

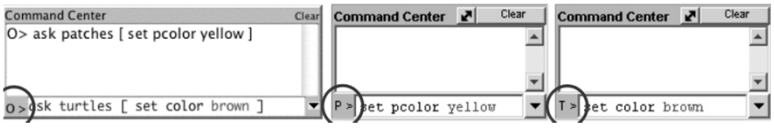
13

# Introduction to NetLogo

## GUI - Controls

- Controls - allow to run and control the flow of execution
  - Buttons
  - Command center
- Buttons - initialize, start, stop, step through the model
  - "Once" buttons execute one action (one step)
  - "Forever" buttons repeat the same action

- Command center - ask observer, patches or turtles to execute specific commands during the execution

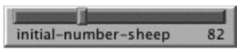



14

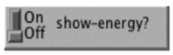
# Introduction to NetLogo

## GUI - Settings

- Settings - allow to modify parameters
  - Sliders
  - Switches
- Sliders - adjust a quantity from min to max by an increment




- Switches - set a Boolean variable (true/false)




- Choosers - select a value from a list

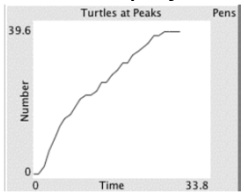
15

# Introduction to NetLogo

## GUI - Views

- Views - allow to display information
  - Monitors
  - Plots
  - Graphics window
- Monitors - display the current value of variables

- Plots - display the history of a variable's value

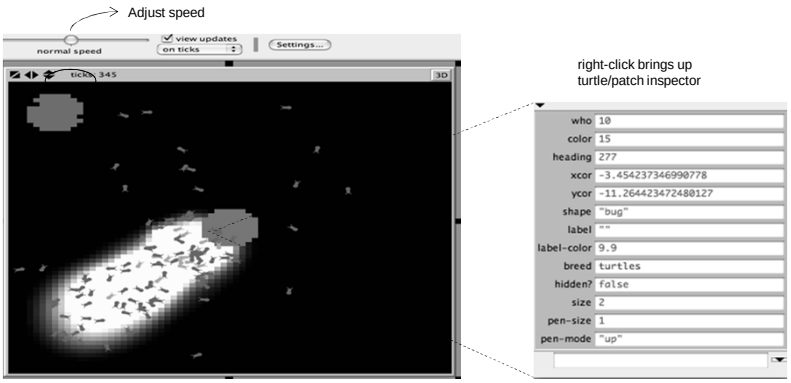



16

# Introduction to NetLogo

## GUI - Views

- Graphics window - The main view of the 2-D NetLogo world




17

# Introduction to NetLogo

## Programming Concepts

- Agents
- Procedures
- Variables
- Ask
- Agentset
- Breeds
- Synchronization

18

# Introduction to NetLogo

## Programming Concepts - Agents

- Each agent can carry out its own activity, all simultaneously
  - Patches
    - Form the 2D world – They don't move, but they sense
    - They have *integer* coordinates (pxcor, pycor)
    - Can generate turtles
  - Turtles
    - move on top of the patches
    - have decimal coordinates (xcor, ycor) and orientation (heading)
  - Observer
    - Can create new turtles
    - Can have read/write access to all the agents and variables

19

# Introduction to NetLogo

## Programming Concepts – Procedures (i)

- **Commands** and **Reporters** tell agents what to do.
- A command is an action for an agent to carry out, resulting in some effect.
- A reporter is instructions for computing a value, which the agent then "reports" to whoever asked it.
- **Procedures**: Commands or Reporters implemented by the user
- **Primitives**: Commands or Reporters built into NetLogo (language keywords)

20

# Introduction to NetLogo

## Programming Concepts – Procedures (ii)

- **Command** executes a finite set of instructions and exists the procedure

```
to setup
   clear all
   create 10
end
```

```
to draw-polygon [ num-sides size ]
   pd repeat num-sides
   [ fd size rt (360 / num-sides) ]
end
```

- **Reporter** computes a result and report it
  - Usually begin with nouns or nouns-phrases

```
to-report absolute-value [ number ]
   ifelse number >= 0
      [ report number ]
      [ report 0 - number ]
end
```

21

# Introduction to NetLogo

## Programming Concepts – Variables (i)

- Variables
  - **Global variables**: there is only one value for the variable, and every agent can access it.
  - **Agent variables:** Each turtle has its own value for every turtle variable. The same goes for patches and links. Think of it as agent properties.
  - **Local variable:** defined and used only in the context of a particular procedure.
    - **scope** of a variable

22

# Introduction to NetLogo

## Programming Concepts – Variables (ii)

- Any variable can typically receive any value type at any given time except global variables that come from GUI components (sliders, etc)
- In NetLogo, variable values can be of the following types:
- **Numbers**: 1, -2, 0.5, -0.1235
  **Strings:** “xpto”, “a”, “123xpto”
  **Boolean:** true, false
  **Agents:** turtle 1, patch 0 0, link 0, one-of turtles
  **Agentsets:** turtles, patches, n-of turtles, n-of links
  **Lists**: [1 2 3 4], list 1 2, (list 1 2 3 4 5 6)

23

# Introduction to NetLogo

## Programming Concepts – Variables (iii)

- Built-in:
  - Turtle variables: **color, xcor, ycor, heading**, etc
  - Patch variables: **pcolor, pxcor, pycor,** etc
- Defining global variables:
  - **globals [ variable-name1 variable-name2]** ; at the beginning of the code
  - adding **a switch, slider, chooser,** or **input box.**
- Define turtle/patch/link variables using the command turtles-own, patches-own or links-own
  - **turtles-own [ energy speed ]**
  - **patches-own [ friction ]**
- Define a local variable using the **let** command
  **let temp** color-of turtle1

24

# Introduction to NetLogo

## Programming Concepts – Variables (iv)

- You can set variable values by using the **set** command: **set variable-name value**
- You can set **global variables** anywhere in the code as these can be accessed by any agent.
- **Local variables** are only accessible inside the procedures or code blocks where they where defined with let.
- **Agent variables** can be read outside an agent with the **of** command: **[color] of one-of turtles.**
- **Agent variables** can only be set by the agents they belong to with **ask** (inside an agent context this is).

25

# Introduction to NetLogo

## Programming Concepts - Ask

- Ask - specify commands to be run by turtles or patches
- Examples
  - asking all turtles:
    - **ask turtles [ fd 50 ... ]**
  - asking one turtle:
    - **ask turtle 5 [ ... ]**
  - asking all patches
    - **ask patches [ diffuse ... ]**
- Only the observer can ask all turtles or all patches

26

# Introduction to NetLogo

## Programming Concepts – Agentsets (i)

- Agentset - definition of a subset of agents
  - Contain either turtles or patches
  - Is in a random order
  - Allows to construct agentsts that contain some turtles or patches
- Example:
- all red turtles:
  - **turtles with [ color = red ]**
- all red turtles on the patch of the current caller (turtle or patch):
  - **turtles-here with [ color = red ]**
- all patches on right side of screen:
  - **patches with [ pxcor > 0 ]**
- all turtles less than 3 patches away from caller (turtle or patch):
  - **turtles in-radius 3**

27

# Introduction to NetLogo

## Programming Concepts – Agentsets (ii)

- Using agentsets
  - ask such agents to execute a command
    - **ask <*agentset*> [ ... ]**
  - check if there are such agents
    - **show any? <*agentset*>**
  - count such agents
    - **show count <*agentset*>**
- example: remove the richest turtle (with the maximum “assets” value)
  - **ask max-one-of turtles [** sum assets **] [ die ]**

28

# Introduction to NetLogo

## Programming Concepts - Breeds

- Breed - a “natural” kind of agentset
  - Different breeds can behave differently
  - breed [wolves wolf]
  - breed [sheep a-sheep]
- A new breed comes with automatically derived primitives:
  - **create-<breed>, create-custom-<breed>, <breed>-here, <breed>-at**
- Breed is a turtle variable
  - **ask** turtle 5 **[ if breed** = sheep ... **]**
- A turtle agent can change breed
  - **ask** turtle 5 **[ set breed** sheep **]**

29

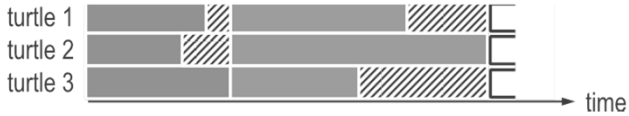
# Introduction to NetLogo

## Programming Concepts - Synchronization

- Agents run in parallel (each agent is an independent thread)
  - asynchronous commands:
    - **ask turtles [ fd random** 10
      do-something**]**

René Doursat, 2008

- Agent threads wait and “join” at the end of a block
  - synchronous commands:
    - **ask turtles [ fd random** 10 **]**
    - **ask turtles [** do-something **]**




René Doursat, 2008

30

# Introduction to NetLogo

## NetLogo Model Skeleton: setup

- Typically you start by defining a button for setting up the initial model state.
- When creating a button you have to assign a command to it. (in this case ***setup***).
- The command associated with the button has to exist in the ***code panel*** and will be executed when the button is pressed.
- Each command is defined by a structure:

to ***command-name***
    *set of instructions*
end

31

# Introduction to NetLogo

## NetLogo Model Skeleton: setup

```
;this procedure is used to setup the simulation initial state
to setup
  clear-all ;resets the world to its initial state
  reset-ticks ;creates or resets the tick counter
  show "cleared"

  ; see Agentset in NetLogo dictionary
  ; ask agentset [commands]
  ; n-of size agentset
  ask n-of population patches[
    sprout 1[
      ;you could set the turtle properties here
      ]
  ]
end
```

32

# Introduction to NetLogo

## NetLogo Model Skeleton: go

- You can then create two more buttons:
  - one **go**/***step*** button that executes one step of the simulation.
  - one ***go*** forever button (a button that executes a given command continuously) that executes the step command forever or until a given condition is met.
- Associate the respective commands to each button.
- **Note:** you can just call step on the go button or add multiple commands to it.

33

# Introduction to NetLogo

## NetLogo Model Skeleton: go

- In your setup command you want to:
  - clear the model components.
  - reset all the variables to their default initial values.
  - create the initial state for your model:
  - The sprout command creates a turtle on the same patch that calls it.
- In the step command you will:
  - ask a group of agents to do something:

```
ask turtles [
  <set of instructions, either existing netlogo commands or newly created commands>
]
```

34

# Introduction to NetLogo

## NetLogo Model Skeleton: go

```
;procedure to determine what happens in a simulation step
to step
  ask turtles [
     ;facexy takes two coordinates in this case, random
     facexy random world-width random world-height
     forward 1
  ]
  tick ; updates the tick count
end

;call step forever (this is called by a forever button)
to go
  step
end
```

35

# Introduction to NetLogo

## Creating model reporters

- Measurements about values you want to extract from a model.
- Examples
  - Percentage of agents that execute a specific behaviour.
  - Number of agents with a given property.
  - Distribution of a particular agent property.
- The reporters can be used by plots and behavior space to export data about your model.

36

# Introduction to NetLogo

## Plotting - 1

- **Plots** can be created using the NetLogo **interface builder**.
- Each plot contains a number of **pens**.
- **Each pen draws** lines, points or bars in the plots and can be configured by editing the plot in your model interface.

37

# Introduction to NetLogo

## Plotting - 2

- Each pen is associated with a command:
  - plot value
  - histogram list
- **Example:**
  - plot count turtles with [color = blue]
  - histogram [age] of agents
- **Note:** using the command **of** with an Agentset (agents, turtles, or patches, etc) results in a list of values.

38

# Introduction to NetLogo

## Plotting - 3

- The plots in a NetLogo model are update
- automatically with the **tick** command.
  This means that each time **tick** is called the plots run the commands in each **pen**.
- To use automatic plotting in your model you need to use the following configuration:

```
to setup
  clear-all
  ...
  reset-ticks
end

to go
  ...
  tick
end
```

39

# Introduction to NetLogo

## Extensions & Tools

- Extensions Guide
- Sound
- Robotics/NetLogoLab /Arduino
- **GIS**
- Bitmap
- Quicktime for Java
- BDI-like architecture
- FIPA-like communication
- Prolog
- Matlab / R
- SQL
- Applets
- Shapes Editor
- Behaviour Space
- System Dynamics
- **HubNet**
- Logging
- Controlling
- Mathematica link
- **NetLogo 3D**

40

# NetLogo References

- NetLogo user manual http://ccl.northwestern.edu/netlogo/docs/
- NetLogo Learning Lab http://www.professorgizzi.org/modelingcomplexity/netlogo/index.html
- NetLogo 4.0 – Quick Guide, Luis R. Izquierdo
- Fundamentals of Multi-agent Systems with NetLogo Examples, José M. Vidal
  http://multiagent.com/p/fundamentals-of-multiagent-systems.html
- Origins of Life: From Geochemistry to the Genetic Code
  http://origins.santafe.edu/tutorials/netlogo

41

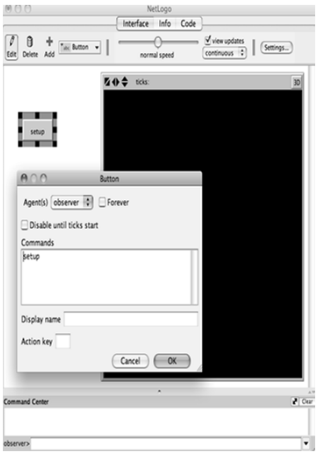
# A simple tutorial

- Create via “File/New”, a new NetLogo program
- Save it, via “File/Save as” with the name *MushroomHunt.nlogo*
- From the “Settings” button
  - view of the World’s geometry
- To initialize the World and run the model
  - setup procedure
  - go prcedure

42

1

- “Interface” tab -> “Button”
- create *setup* button
- similarly create a *go* button




43

2

- In “Code” tab
  - Create the skeleton of setup & go

```
to setup
  ca
  reset-ticks
end

to go
end
```

  - Change setup to

```
to setup
  ask patches
  [
    set pcolor red
  ]
end
```

  - Create the clusters of mushrooms (patches).
    - The cluster can be a model parameter
    - Define a global variable *num-clusters*

```
globals [num-clusters]
```

  - Modify the setup to turn in red randomly a “num-cluster” patches

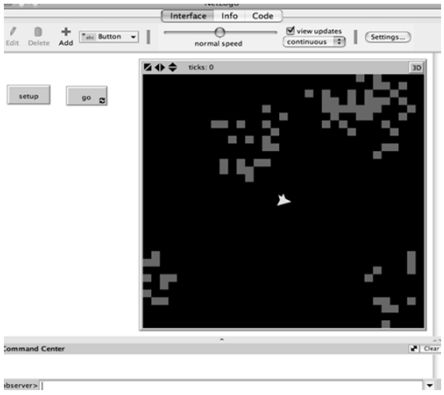
```
to setup
  ca
  ask n-of num-clusters patches
  [
    ask n-of 20 patches in-radius 5
      [
       set pcolor red
      ]
  ]
  reset-ticks
end
```

44

## 3

- create the turtles
  - use the primitive create-turtles

```
create-turtles 2
[
  set size 2
  set color yellow
]
```




45

## 4

- In the go procedure
  - Tell to turtles what to do. In this case to search for mushrooms
  - So we need a search procedure

```
to go
  ask turtles [search]

end

to search
end
```

  - Let's define search.

```
to search
  ifelse time-since-last-found <= 20
    [right (random 181) - 90]
    [right (random 21) - 10]

  forward 1

end
```

  - After globals statement define

```
globals [num-clusters]
turtles-own [time-since-last-found]
```

46

## 5

- We update the setup procedure

```
to setup
  ca
  set num-clusters 4
  ask n-of num-clusters patches
  [
    ask n-of 20 patches in-radius 5
     [
      set pcolor red
     ]
  ]

  create-turtles 2
  [
    set size 2
    set color yellow
    set time-since-last-found 999
  ]
  reset-ticks
end
```

47

## 6

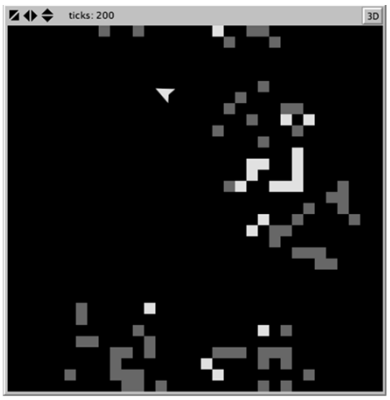
- and the search procedure as well as

```
to search
  ifelse time-since-last-found <= 20
    [right (random 181) - 90]
    [right (random 21) - 10]

  forward 1

  ifelse pcolor = red
    [
      set time-since-last-found 0
      set pcolor yellow
    ]
    [set time-since-last-found time-since-last-found + 1]

end
```

48

7

```
globals [num-clusters]
turtles-own [time-since-last-found]

to setup
  ca
  set num-clusters 4
  ask n-of num-clusters patches
  [
    ask n-of 20 patches in-radius 5
     [
      set pcolor red
     ]
  ]

  create-turtles 1
  [
    set size 2
    set color yellow
    set time-since-last-found 999
  ]
  reset-ticks
end

to go
  tick
  ask turtles [search]
end

to search
  ifelse time-since-last-found <= 20
    [right (random 181) - 90]
    [right (random 21) - 10]

  forward 1

  ifelse pcolor = red
    [
      set time-since-last-found 0
      set pcolor yellow
    ]
    [set time-since-last-found time-since-last-found + 1]

end
```




49

The modelling cycle for the Mushroom-hunter problem

1. Formulate the problem
   - What search strategy maximizes the rate of finding items if are distributed in clusters?
2. Formulate hypothesis for essential processes and structures
   - process switches from large-scale movements to small-scale searching depending on previous
   - discoveries
3. Choose scales, entities, state variables, processes and parameters
4. Implement the model
5. Analyse, test and revise the model
   - we could the model by trying different search algorithms and parameter values analyse to see which produces the highest rates

50